INSTITUTO DE MATEMÁTICA E ESTATÍSTICA - IME
UNIVERSIDADE DE SÃO PAULO - USP
Introdução à Programação Concorrente, Paralela e Distribuída

EP3: Cálculo Distribuído do Conjunto de Mandelbrot

Alunos: Fábio Eduardo Kaspar N. Usp: 7991166
Ricardo de Oliveira N. Usp: 3683165
Ricardo Oliveira Teles N. Usp: 7991444

1. Introdução

Neste Exercício Programa (EP3), iremos avaliar o desempenho do algoritmo para o cálculo do Conjunto de Mandelbrot para sistemas distribuídos. Este algoritmo, estudado no EP1 para sistemas paralelos, apresentou um grande ganho de desempenho quando executado utilizando números de threads igual ao número de cores da máquina. O Conjunto de Mandelbrot foi descoberto "por Benoit Mandelbrot, que trabalhava na IBM durante a década de 1960, e foi um dos primeiros a usar computação gráfica para mostrar como a complexidade pode surgir a partir de regras simples" (Enunciado do EP3).

2. Implementação

Este EP foi implementado na linguagem C e usou-se a biblioteca OpenMPI para distribuir o código nas máquinas e OpenMP para paralelizar os códigos entre as threads. Foram implementadas três versões para o mesmo cálculo do Conjunto de Mandelbrot a fim de mensurar qual é o ganho de desempenho da implementação distribuída em relação à implementação sequencial. E a implementação distribuída e paralelizada teve como objetivo avaliar e mensurar o desempenho do cálculo com a variação do número de máquinas versus número de cores, tendo em vista que conforme foi se aumentando o número de máquinas foi diminuindo na mesma proporção o número de cores por máquina.

A implementação sequencial (mandelbrot_seq.c) adotada foi a versão disponibilizada no EP1 pelo monitor da disciplina. Ela se utiliza de três ***for*** aninhados. Dois deles percorrem as coordenadas x e y da imagem e o terceiro ***for***, mais interno, itera sobre o ponto.

A segunda implementação (mandelbrotOpenMPI_seq.c), desenvolvida pelo grupo, é um código distribuído e executado de forma sequencial dentro das máquinas, isto é, não faz o uso de threads. Calcula-se o tamanho total da imagem e divide a quantidade de pixels igualmente entre as máquinas, as quais computam e armazenam num vetor o número de iterações do Conjunto de Mandelbrot sobre cada pixel. Por fim, cada máquina envia o seu vetor de iterações para a máquina root (rank = 0) através da função MPI_Gatherv, que se responsabiliza em computar a cor de cada pixel (com referência no número de iterações) e construir a imagem final. O parâmetro, número de processos (np), passado ao programa durante sua chamada é igual ao número de máquinas disponíveis em cada experimento: 1, 2, 4 e 8 instâncias, respectivamente.

Já a última implementação (mandelbrotOpenMPI_omp.c) é um código distribuído, que faz uso de threads aplicando as diretivas de compilador fornecidas pelo OpenMP. Nesta implementação, é adotado o mesmo processo de divisão de tarefas para as máquinas e cada uma delas divide o seu vetor de computação entre as threads. A escolha da quantidade de threads adotada, em cada um dos 4 experimentos, é baseada no número de cores das máquinas instanciadas para o experimento, ou seja, número de threads igual ao número de cores, conforme conclusão que se chegou com o EP1 (melhor desempenho).

Para reprodução deste experimento rode o <Makefile> em cada máquina, configure o SSH das máquinas e gere um arquivo com nome <hosfile> contendo a lista dos IPs das máquinas que irão receber o código distribuído. Após, rode o <run_measurements.sh>. Finalmente rode o <gera_boxplot.py> com o python para gerar os gráficos.

Estes experimentos foram realizados na Google Compute Engine (GCE) com as seguintes configurações de sistemas: conjuntos de 1, 2, 4 e 8 instâncias com 8, 4, 2 e 1 cores, respectivamente. Foram geradas imagens com tamanho de entrada N = 8192 para as quatro regiões exploradas no EP1. Foram processadas e analisadas versões COM e SEM I/O e alocação de memória para se avaliar também o seu impacto (análise extra - não exigida no EP).

3. Análise dos resultados

3.1. Implementação sequencial

Nos dados obtidos com a versão sequencial podemos observar que, independentemente do número de cores da máquina (exceto para 4 máquinas com 2 cores cada), o desempenho de processamento é aproximadamente igual em todos os casos. E a diferença das versões COM ou SEM I/O e alocação de memória é um valor constante (aproximadamente 7 segundos).

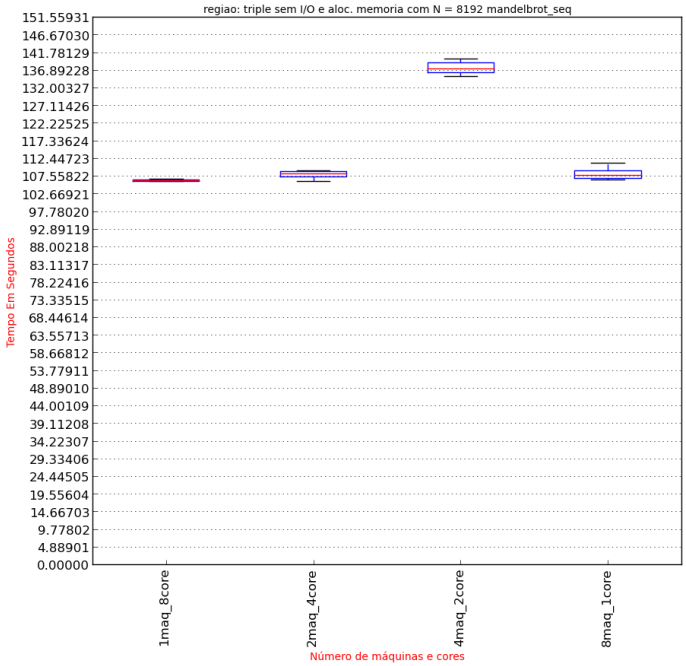

Figura 1

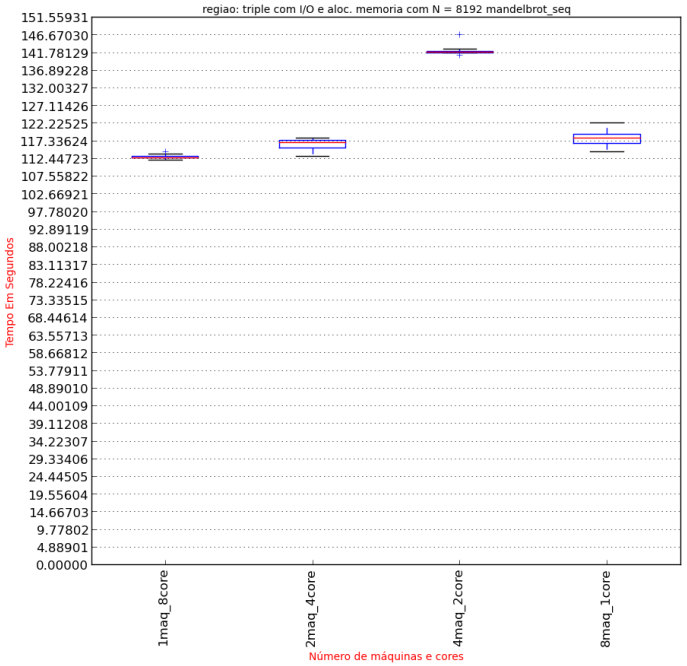

Figura 2

3.2. Implementação distribuída

Já para os dados obtidos com a implementação distribuída percebe-se o ganho de desempenho proporcional ao número de máquinas que processam o código distribuído. Portanto, dobrando-se o número de máquinas o processamento é realizado na metade do tempo. E novamente a diferença entre as versões COM ou SEM I/O e alocação de memória é um valor constante.

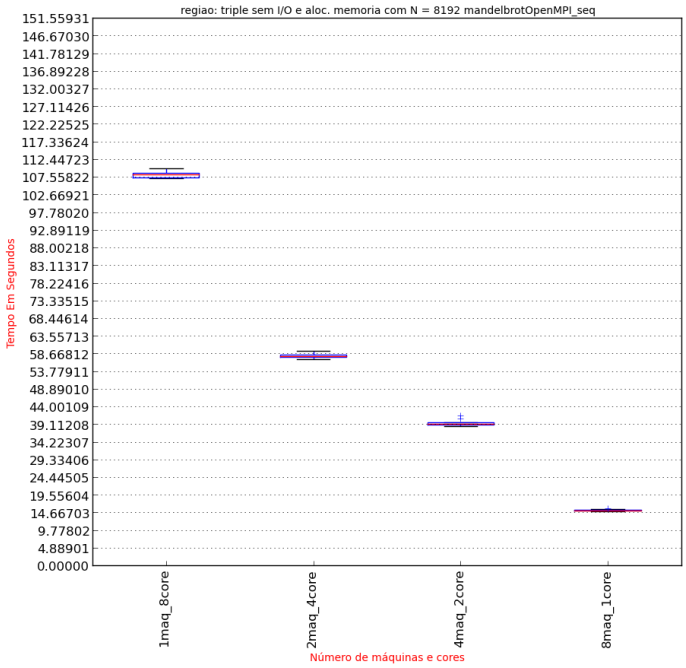

Figura 3

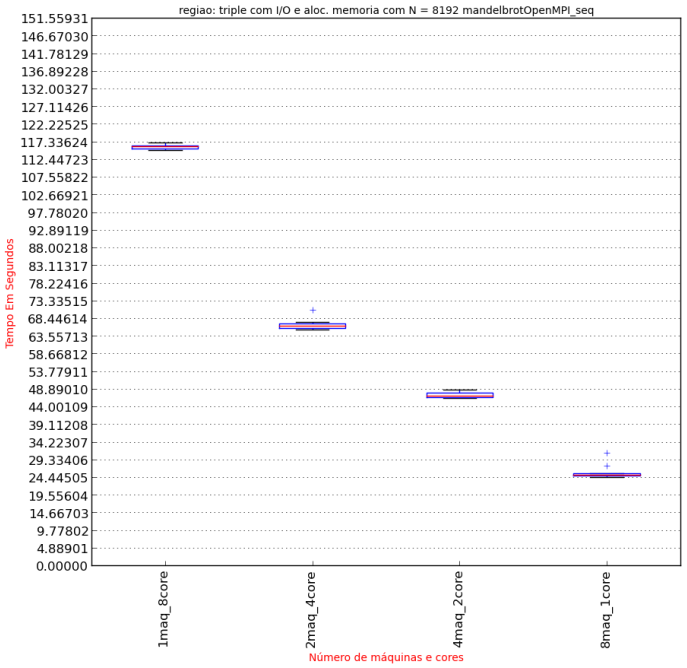

Figura 4

3.3. Implementação distribuída e paralelizada

O resultado mais significativo foi observado para a implementação distribuida e paralelizada devido ao tipo de ambiente específico em que esses testes foram obtidos. Nesse ambiente conforme aumenta-se o número de máquinas diminui-se o número de cores por máquinas na mesma proporção. Portanto, o resultado observado nos permite afirmar que o aumento do número de máquinas para distribuição de processamento tende a compensar a diminuição do número de cores para a paralelização do processamento. Desta forma, o resultado obtido revelou que o desempenho permanece constante e alto em relação às demais implementações.

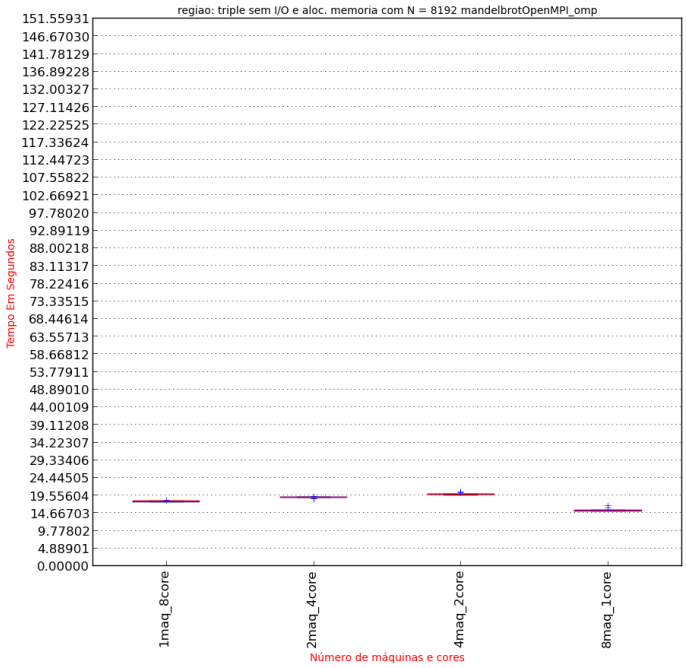

Figura 5

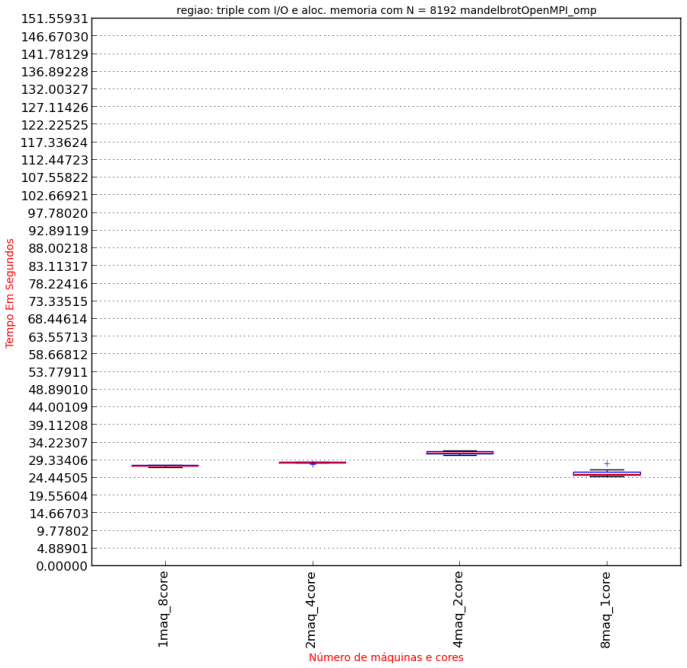

Figura 6

3.4. Comparação entre as Implementações

Os resultados produzidos revelaram que a implementação mandelbrotOpenMPI_seq, quando temos somente uma máquina instânciada, tem resultados de desempenho muito próximos dos resultados obtidos pela implementação mandelbrot_seq. Isto nos mostra que uma máquina com número de processos igual a 1 acaba resultando num modelo sequencial de implementação. No entanto, percebe-se claramente que o tempo de execução cai exponencialmente conforme cresce exponencialmente o número de máquinas instânciadas, conforme é observado nos dados. Portanto, temos que o ganho de desempenho é diretamente proporcional ao número de máquinas que distribuem o processamento.

Outra observação importante a se fazer é que, na implementação mandelbrotOpenMPI_omp, o desempenho se mantém praticamente constante conforme aumenta o número de máquinas, aparentemente não mostrando ganho de rendimento. Entretanto, observa-se que o desempenho é aproximadamente igual ao melhor desempenho da implementação mandelbrotOpenMPI_seq (com oito máquinas processando). Este resultado ocorre pelo fato de que conforme aumenta o número de máquinas, diminui o número de cores por máquina, na mesma proporção.

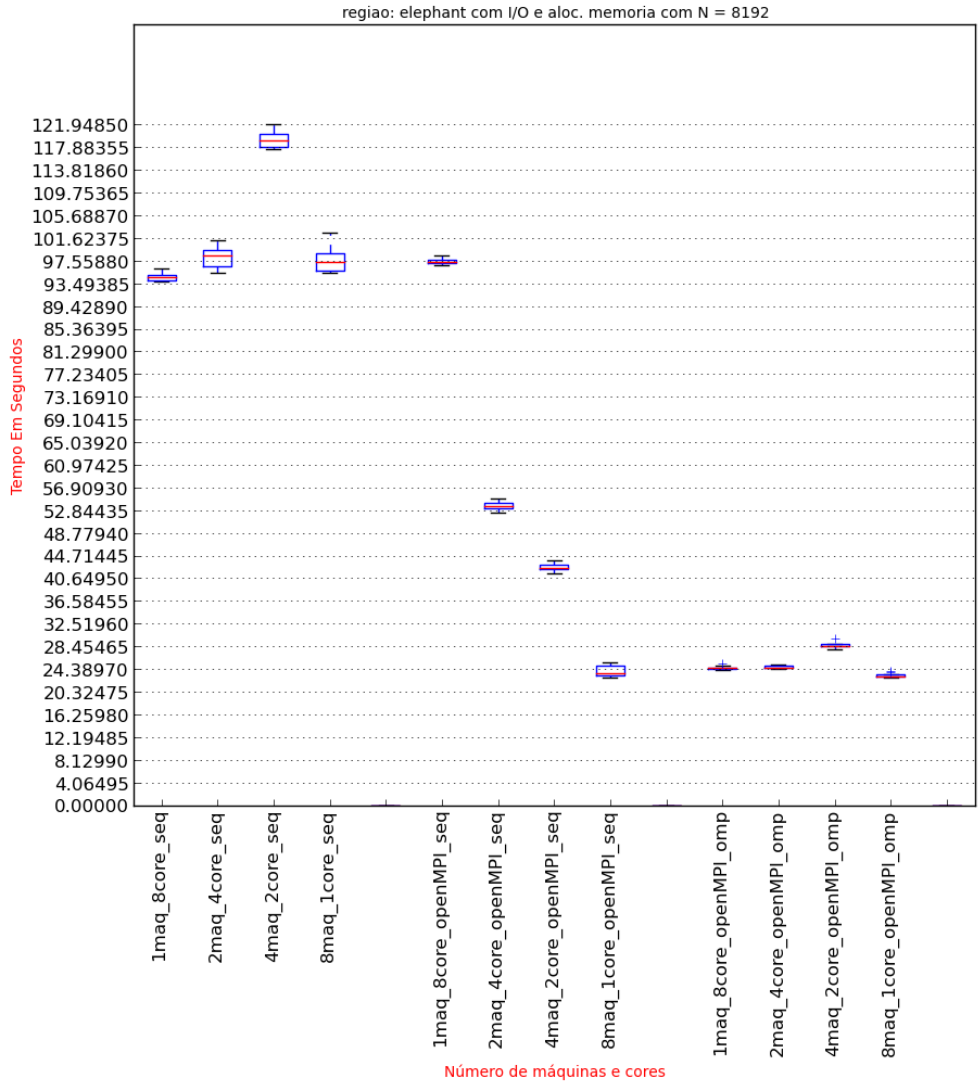

Figura 7

Outro aspecto interessante a se observar nesses experimentos é que o real ganho de desempenho pelos processos distribuídos e pelos processos distribuídos/paralelizados só começam a ser realmente significativos quando se tem um alto grau de processamento. Isso pode ser notado comparando os resultados entre duas regiões distintas do Conjunto Mandelbrot como, por exemplo, comparando a região Full que levou em média 17 segundos de processamento com a região Triple_espiral que chegou a levar por volta de 105 segundos para implementação sequencial.

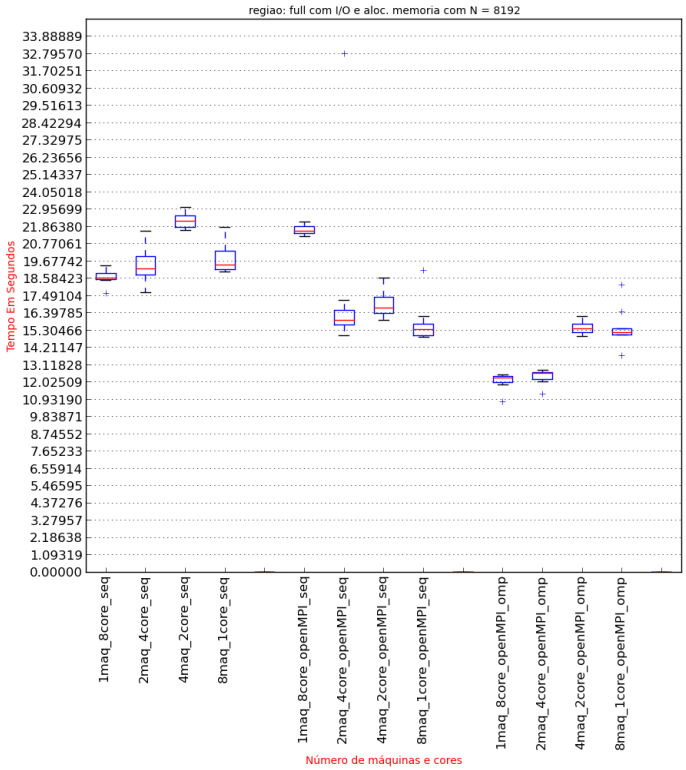

Figura 8

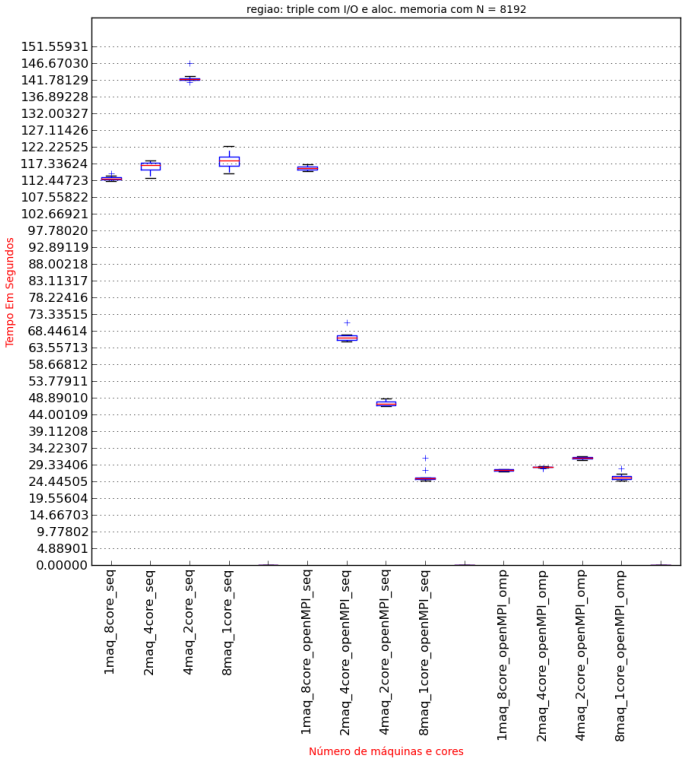

Figura 9

Obs: Para visualizar os demais gráficos, acesse:

`https://github.com/ricardoteles/ep3-219.git`

4. Conclusão

A computação paralela e distribuída apresentam vantagens em algoritmos, cujo o processamento é maior que operações de I/O e alocação de memória, uma vez que essas são realizadas de modo sequencial. Esse experimento evidenciou que o uso da computação distribuída, aplicada em máquinas com poucos cores, permite alcançar o mesmo ganho de desempenho da computação paralela, em uma máquina com muitos cores. No entanto, quando se tem disponível muitas máquinas com vários cores, a melhor implementação a ser adotada é a combinação entre a programação paralela e a distribuída.